09 | 容量预测（下）：为不同服务“画像”，提升容量预测准确性

2021-05-31 吴骏龙

《容量保障核心技术与实战》 课程介绍 >

讲述：吴骏龙
时长 15:34 大小 14.27M

你好，我是吴骏龙。

在容量预测的上篇中，我介绍了容量预测的基本过程和方法，在课后讨论环节也出了一道题目供你思考，你都学会了吗？掌握了这些基础知识后，在这一讲我们继续讨论容量预测中的一些进阶问题，这些问题都是我在实践中摸爬滚打提炼出来的，如果你想在工作中应用容量预测，那么这些问题也是绕不开的。

首先，虽然我们已经建立了理想化的模型，但现实情况是，这个模型可能针对某些服务的预测结果总是不那么准确，而且无论怎么调整模型的参数都收效甚微。这时候，我们应回过头思考一下，当初选择的特征是否合适？

其次，服务是在不断迭代的，不断有新功能上线，这就意味着服务的容量始终处于变化的过程中，如果容量预测的模型也跟着高频变化，计算量就会非常大，怎么权衡好服务迭代和模型更新之间的关系呢？

最后，即便服务不变，业务场景的变化也会造成服务容量的变化，例如大促活动带来局部几个服务的流量突增，如果我们按照非大促期间业务场景的流量特征去建立模型，对容量进行预测，肯定是不准确的，那该如何应对呢？

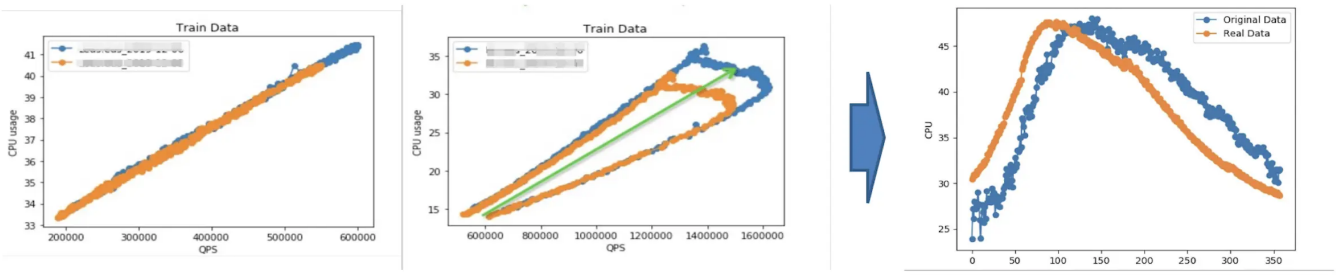
别急，上面这三个问题，都是有解决方案的。我们赶紧来看一看这些问题都是如何解决的吧。

相关度分析与服务画像

在上一讲所提到的特征选取过程中，我比较“草率”地将服务的 TPS 和依赖服务的 TPS 作为输入，与 CPU 利用率建立模型。但实际情况下，**CPU 利用率可能不仅仅受 TPS 制约**，如果我们忽略其他特征的话，有很大可能就会影响模型的准确性。

下图展示了 TPS 和 CPU 利用率的可视化映射关系，通过左图可以看到，两者的关联是非常紧密的，这说明除 TPS 以外，其他因素对 CPU 利用率几乎没有什么影响，这是比较理想的情况；而右图则出现了不紧密的映射关系，甚至一个 TPS 会对应多个 CPU 利用率，很明显还有我们没有考虑到的因素在影响着 CPU 利用率，这会导致模型拟合不佳，那么如何解决这个问题呢？

TPS与CPU的映射关系　　拟合不佳



首先，我们要能够对 TPS 和 CPU 利用率的相关度做定量分析，并根据相关度制定不同的应对策略，这是一切后续工作的基础。

那怎么做相关度的定量分析呢？在统计学中，**皮尔逊相关系数**（Pearson product-moment correlation coefficient）就是用于度量两个变量 X 和 Y 之间的相关程度的，其值介于 -1 和 1 之间。

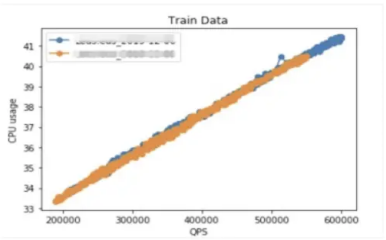
$$r = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \overline{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \overline{Y})^2}}.$$

我们可以对 TPS 和 CPU 利用率这两个变量计算皮尔逊相关系数，再进行分类：

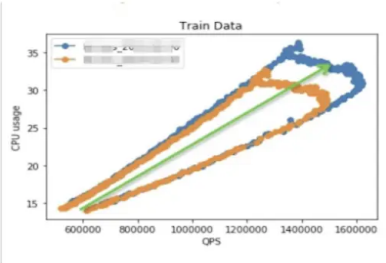
皮尔逊相关系数的绝对值介于 0.5 - 1 之间，为强相关

皮尔逊相关系数的绝对值介于 0.1 - 0.5 之间，为弱相关

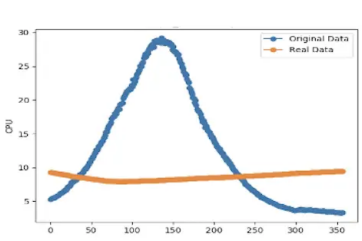
皮尔逊相关系数的绝对值介于 0 - 0.1 之间，为不相关



r ≈ 0.99
强相关



r ≈ 0.49
弱相关



r ≈ 0.08
不相关

针对强相关的场景，意味着 CPU 利用率几乎只受 TPS 的影响，这时直接建立 TPS 和 CPU 利用率的模型就可以比较好的拟合出结果了。而不相关的场景，一般是一些批处理服务或任务型的服务，这些服务没有外部流量，可以特殊处理或直接过滤掉。我们重点来看一下弱相关的场景如何处理。

弱相关的场景下，CPU 利用率不仅仅受 TPS 制约，还伴有其他因素。我提供两种思路解决这个问题：第一，找出所有影响 CPU 利用率的特征，**对服务进行画像**，即针对每个服务选

取不同的特征去表示它；第二，建立一个 TPS 映射 CPU 利用率的**概率表**，选取出现概率最大的 CPU 利用率值作为特征。下面我具体展开这两种思路的做法。

基于第一种服务画像的思路，我们需要考虑更多特征，从中选取对 CPU 利用率影响最大的一个 / 些去建模，或筛去一些无关的特征不去建模，以下是几种常见的方式：

1. 目测法： 将所有特征做归类，如果一个特征能从另一个特征推导出来，我们就叫它多余特征，无需考虑。如下图中，我们假设理想情况下内存、带宽、连接数等指标的变化最终都会反映到服务 TPS 上，那么服务 TPS 就称之为有效特征，其他指标就是多余特征。应用 Owner 是谁对服务容量没有影响，称为无关特征。我们最终只需要考虑有效特征。

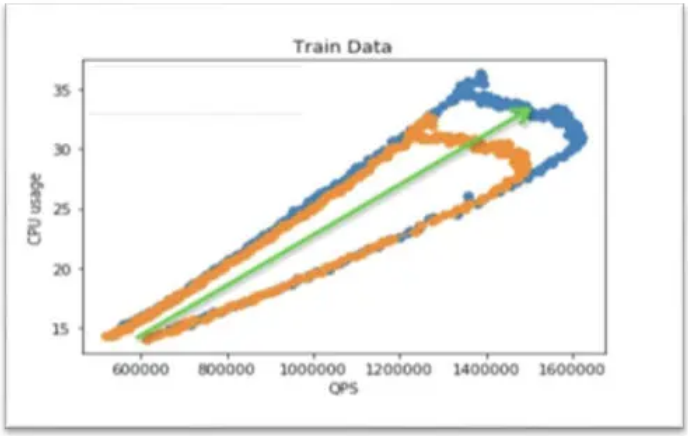
2. 过滤法： 去掉取值变化小或不变的特征，这些特征虽然对容量有影响，但影响的程度保持不变，所以可以约简。比如某服务的成功率长期处于 100%，那么“成功率”这个特征就不需要考虑。

3. 包裹法： 每次选择若干特征或者排除若干特征，进行模型评优，直到选择出最佳的子集。这有点像是排列组合，去寻找一个最佳的特征组合，最准确的预测 CPU 利用率。

4. 嵌入法： 先使用某些机器学习的算法和模型进行训练，得到各个特征的权值系数，根据系数从大到小选择特征。类似于过滤法，只不过是通过训练来确定特征的优劣。Spark 中的 featureImportance 就是一个典型的嵌入法实现。

服务画像的关键点是，对 CPU 利用率影响较大的特征必须都包含在我们的选择范围内，否则只能是巧妇难为无米之炊，但有时候我们确实很难找全所有的特征，这时候可以换第二种思路，通过建立概率表的方式去解决问题。

具体做法是，建立一个 TPS 映射 CPU 利用率的概率表，根据概率表找出某个 TPS 区间内，CPU 利用率出现频率最高的值，供建模使用。下图展示了一个概率表的实例，通过左下角的图可以看到，TPS 处于某水位时 CPU 利用率的出现频率分布有多个极大值，但肯定**只会有一个最大值**，这就是我们要找的值；通过右边的表格能够更直观的看到，标注为红色的行就是对应 TPS 下的 CPU 利用率出现次数最多的值，在建立模型时只需要考虑这几行的数据即可。



样本数据片段

TPS	CPU	COUNT
600000	12	56
600000	15.5	298
700000	16	74
700000	18	216
800000	17.8	68
800000	18	31
800000	20	329

概率表方式背后的思想是，出现频次最高的数据从概率角度往往是更可靠的，也最能够代表实际情况。

服务画像和概率表的方式各有利弊，当我们能够寻找到所有准确的特征时，服务画像的准确性是更高的；反之，可以通过概率表的方式曲线救国。

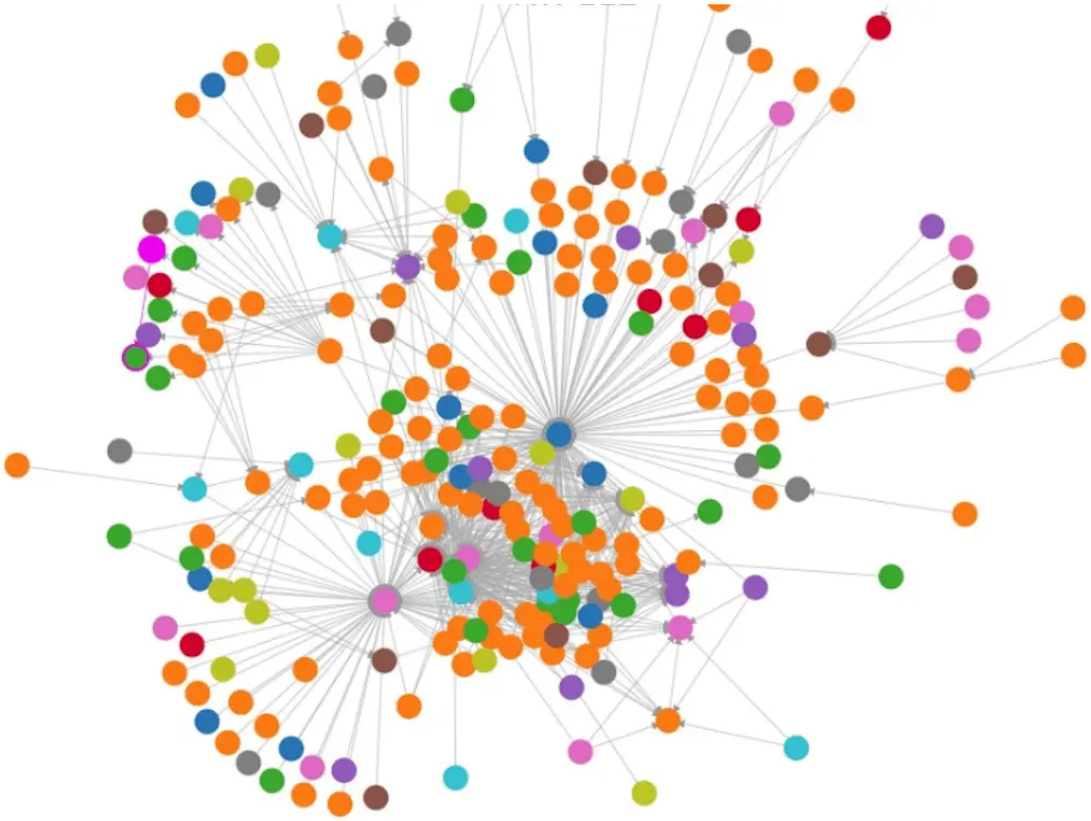
容量预测迭代与校准

解决了容量预测准确性的问题，我们进入下一个问题，前面谈到过，**互联网服务是不断迭代向前的，这也就意味着服务画像是不断变化的**，很有可能一次发布就会导致之前的预测结果失效，这就要求容量预测也要进行迭代，及时校准。

容量预测迭代和校准的一个困难点在于，由于服务变更前后的容量并不是递进的关系，因此我们很难采取增量的方式进行学习，大多数情况下还是得重新建模。一种看似比较简单的做法是，在服务发生变更（发布、修改配置、扩缩容等）后，对该服务重新进行一次建模，直接覆盖掉上一次结果。

到这里，你应该能够意识到这种方式有一个很大的弊端，即它没有考虑服务的变更对其他服务容量的影响，仅仅对该服务重新进行建模，没有解决其他服务的容量预测准确性问题。

我们依然顺着朴素的思想演进，那是不是可以在任一服务变更后，就对该服务上下游链路所涉及的所有服务都重新建模一遍呢？这种做法在一定程度上是可行的，前提是我们得依赖链路追踪系统找出服务依赖的网状结构，对于依赖方较多的服务，如下图所示，这个网状结构可能会很复杂，计算量依然很大。



在实践中，我们考虑换一种思路，**在某个时间窗口内对所有服务重新建模，而不是一有服务变更就这么做**，这个窗口期可以跟随服务发布期以滑动窗口的形式设置，窗口的大小根据容量结果的时效性进行设置。比如说，我们规定每周三为服务发布期，容量时效性为 3 天，可以设置一个跨度为 3 天的滑动窗口，在第 4 天需要再进行一次重新建模，窗口每次向前移动一周。

这套模式在阿里本地生活经过近 1 年的实践，很好的兼顾了服务迭代周期和预测模型更新频率，也没有过多牺牲容量预测的时效性，同时，扩展性比较好，例如服务在每周有两个发布窗口，那就设置两个滑动窗口就可以了。

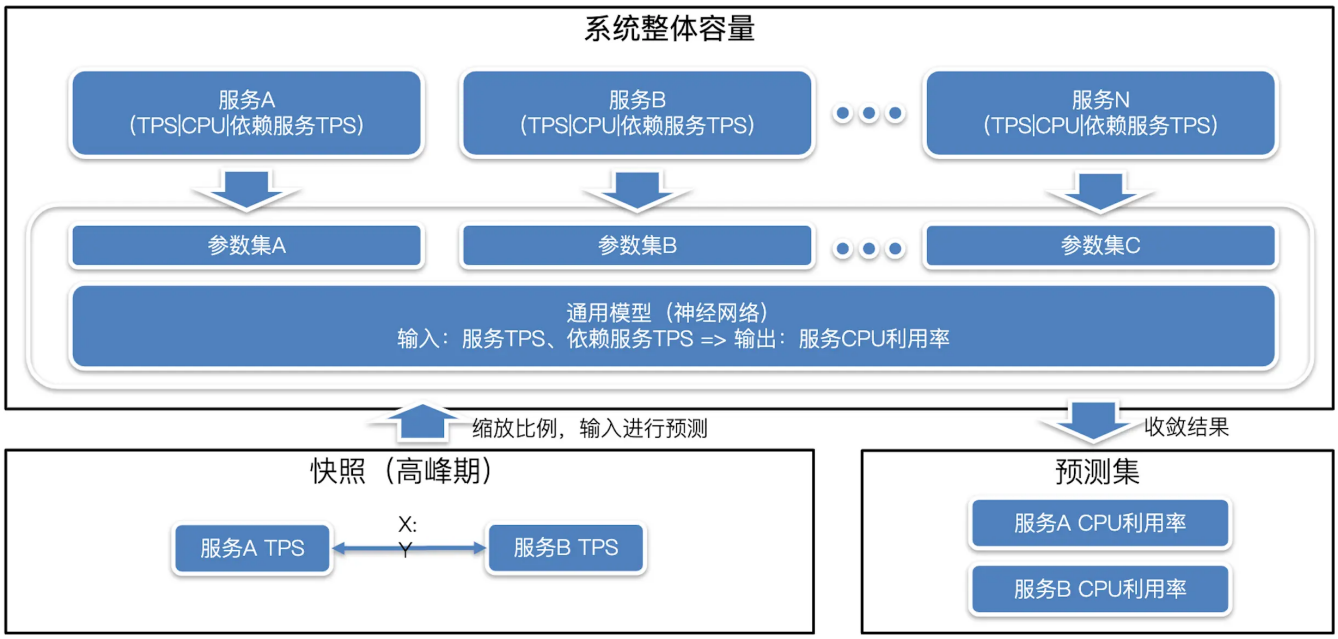
警惕业务场景变化

上面我谈到了服务迭代对容量预测的影响，但它并不是唯一的影响因素，业务场景的变化也会导致容量预测失真。

考虑容量风险会随流量增高而放大，在进行容量预测时，我们一般会使用高峰期的数据进行建模，主要针对的也是高峰期的业务场景。这里就引出了一个问题，如果在未来某段时间内会有一些业务场景发生变化，比如下个月会进行一场大促活动，那么如何预测那时的服务容量呢？我们现在构建的模型，完全有可能不适合未来的场景。

解决这一问题的关键在于，如何获得未来场景的特征数据，以构建出逼近真实情况的模型，来进行预测。 问题的解法就是我之前讲到的全链路压测，通过对特定场景进行压测的方式，获取在高负载的情况下服务的各项指标数据，通过这些数据来构建模型。

当然，全链路压测也不是魔法，不可能做到与真实场景一模一样，为了尽可能减少差异，我们可以通过容量预测和全链路压测双向校准的方式，进行对齐，具体做法相对会复杂一些，请你参阅下图，并集中精力听我讲述。



针对容量预测对全链路压测的校准，需要在**常态业务场景**下对**系统整体容量**进行预测，方法是先在业务高峰期选取一个时间点，获取这个时间点下所有服务的 TPS，它们之间的比例关系称之为 **“快照”**。

接下去，我们可以在保持比例不变的情况下，不断增加每个服务的 TPS 值，并将其输入每个服务的模型进行预测，得到各自的 CPU 利用率，直到某一个关键服务的预测 CPU 利用率超过阈值（如 90%），这时就触及了整体系统能承载的最大容量，我们将所有服务对应的预测 CPU 利用率由高到低排序输出，就能得到一份高危服务的列表，如下表所示。

服务名	高峰期峰值TPS	输入模型的TPS	预测CPU利用率
服务a	87	217.5	97.8%
服务b	109.1	272.75	89.22%
服务c	49.8	124.5	87.73%
服务e	220.75	551.88	82.9%
服务f	18.6	46.5	79.18%
服务g	58.31	145.78	77.06%

将这份列表中各服务的预测 CPU 利用率，与全链路压测达到相同 TPS 下的 CPU 利用率进行比对，若差距过大，则检查全链路压测场景是否有失真，这就完成了容量预测校准全链路压测的工作。

在保证常态业务场景的全链路压测模型无误后，加入或修改有变化的业务场景，更新全链路压测脚本后再次压测，压测过程中的数据指标输出给模型重新学习，这样就完成了全链路压测对容量预测的校准。在服务上线后，根据线上的真实数据，可以再进行几次模型的校准工作，就能达到良好的效果了。

总结一下，俗话说“唯一不变的是变化”，业务场景的变化是必然的结果，要迎合这些变化，保证容量预测的准确性，我们应该对预测模型定期进行校准。全链路压测是一个不错的数据校准的输入源，尤其针对未来发生的业务场景，可能还是唯一的数据源；同样的，容量预测的结果也可以反馈给全链路压测，检验压测模型的准确性，达到双向校准的效果。

总结

今天，我和你讨论了三个典型的容量预测中可能会遇到的高阶问题，针对每个问题，我都给出了不少实践方案，这些方案是经过血与火洗礼的，经过了成百上千个各式各样的服务的验证，不断改进和优化后才呈现出现在的效果，希望它们能帮助你少走一些弯路。

我首先给出了“皮尔逊相关系数”这个工具，对服务 TPS 和 CPU 利用率之间的相关度进行了定量分析，根据相关度的强弱，分别采取不同策略。其中，重点讲到了在两者弱相关时的应对策略，如果能够穷举出尽可能多的相关特征，可以通过特征选取的方式对服务进行画像，提升预测准确率；如果特征非常难找，那么可以依靠概率表的方式曲线救国。

随着服务不断迭代，容量也在不断变化，我与你分析的第二个问题，就是如何平衡好服务迭代和容量预测频率的关系。根据服务发布窗口（或其他变更时间点）建立滑动窗口机制，既保证了在服务变更后能够尽快地更新模型，又不至于带来大量的计算量，是一个不错的实践方式。

业务场景变化也会导致容量变化，针对这个问题，我结合之前提到的全链路压测工作，通过建立全链路压测和容量预测双向校准的机制，提前对变化的业务场景进行预测，识别容量风险。

到这里，容量预测的所有内容就讲完了，纸上得来终觉浅，希望你可以多多实践，遇到困难时可以再回顾一下这两讲的内容，相信你会有更多收获。

课后讨论

在“业务场景变化”这部分的讲解中，我提到了系统整体容量预测的过程，在建立快照后，保持比例不变并不断增加每个服务的 TPS 值输入模型进行预测，直到其中有服务的 CPU 利用率达到瓶颈，就认为是系统整体容量的瓶颈点。

这个过程可以理解为不断向上试探的过程，不过如果增幅太大，可能会出现同时有很多服务的 CPU 利用率预测结果超过阈值，这时对应的系统整体容量已经远远超过了瓶颈点；而增幅太小的话，需要预测的轮次又太多，计算量大且耗时长。请你想一想，有没有更好的策略能快速逼近结果？欢迎与我分享你的思路。

分享给需要的人，Ta订阅后你可得 16 元现金奖励

赞 1 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 08 | 容量预测（上）：第三只眼，通过AI预测服务容量瓶颈

下一篇 10 | 浅谈排队论：数学之美，通过建模计算容量

精选留言 (2)

写留言

P.T
2021-07-20

如果获得了cpu利用率，怎么指导在全链路上的容量缩阔呢？cpu利用率和容量也不一定是线性的吧。

feihui
2021-06-21

有点像如何控制步长的问题，开始使用随机值，然后使用二分法快速收敛

作者回复: 回答的很好！此外，一次预测完成后，下一次就可以使用前一次的结果作为初始值替换掉随机值，收敛更快。